Best Practices for a Handwritten Text Recognition System

George Retsinas[1(✉)], Giorgos Sfikas[2,3,4], Basilis Gatos[2], and Christophoros Nikou[3]

[1] School of Electrical and Computer Engineering, National Technical University of Athens, Athens, Greece
gretsinas@central.ntua.gr

[2] Computational Intelligence Laboratory, Institute of Informatics and Telecommunications, National Center for Scientific Research "Demokritos", Athens, Greece
bgat@iit.demokritos.gr

[3] Department of Computer Science and Engineering, University of Ioannina, Ioannina, Greece
cnikou@cse.uoi.gr

[4] Department of Surveying and Geoinformatics Engineering, University of West Attica, Athens, Greece
gsfikas@uniwa.gr

Abstract. Handwritten text recognition has been developed rapidly in the recent years, following the rise of deep learning and its applications. Though deep learning methods provide notable boost in performance concerning text recognition, non-trivial deviation in performance can be detected even when small pre-processing or architectural/optimization elements are changed. This work follows a "best practice" rationale; highlight simple yet effective empirical practices that can further help training and provide well-performing handwritten text recognition systems. Specifically, we considered three basic aspects of a deep HTR system and we proposed simple yet effective solutions: 1) retain the aspect ratio of the images in the preprocessing step, 2) use max-pooling for converting the 3D feature map of CNN output into a sequence of features and 3) assist the training procedure via an additional CTC loss which acts as a shortcut on the max-pooled sequential features. Using these proposed simple modifications, one can attain close to state-of-the-art results, while considering a basic convolutional-recurrent (CNN+LSTM) architecture, for both IAM and RIMES datasets. Code is available at https://github.com/georgeretsi/HTR-best-practices/.

Keywords: Handwritten text recognition · Convolution - recurrent neural network · Best practices

1 Introduction

Handwritten Text Recognition (HTR) is an active area of research, combining ideas from both computer vision and natural language processing. Unlike recognition of machine-printed text, handwriting is related to a number of unique

S. Uchida et al. (Eds.): DAS 2022, LNCS 13237, pp. 247–259, 2022.
https://doi.org/10.1007/978-3-031-06555-2_17

characteristics that make the task much more challenging than traditional optical character recognition (OCR). The challenging nature of handwriting recognition stems mostly from the potentially high writing variability between individuals. To this end, along with visually decoding an image into sequence of characters, several HTR works adopt language models to reduce this innate ambiguity of handwritten characters, making use of contextual and semantic information.

In general, designing an effective and generalizable learning system is a ongoing challenge, with transferability between different learned writing styles more being not a given in most cases [22]. Neural Networks (NNs), among a variety of other learning systems, have been used for the recognition of handwriting from early on, with a span ranging between simpler subtasks such as single digit recognition [1] up to full, unconstrained offline HTR [7,21]. Following the rise of deep learning and its applications, recent developments in HTR are monopolized by Deep Neural Networks (DNNs). The seminal work of Graves et al. [9] played a pivotal role in the rise of deep learning for HTR applications by enabling the training of recurrent nets without assuming any prior character segmentation. A plethora of subsequent works for HTR relied on Graves et al. in order to train modern and notably effective DNNs [14,16,21,24].

This work focuses on finding best practices for building modern HTR systems. We explore a set of guidelines for training HTR DNNs, re-examining and extending ideas from several previous works of ours [23–25]. We start with a fairly common deep network architecture for HTR, consisting of a CNN backbone and a BiLSTM head, and we make simple yet effective architectural and training choices. These best practice suggestions can be categorized and summarized as follows:

1. **pre-processing:** retain aspect ratio of images and use batches of padded images in order to effectively use mini-batch Stochastic Gradient Descent (SGD)
2. **architectural:** replace the column-wise concatenation step between the CNN backbone and the recurrent head with a max-pooling step. Such a choice not only reduces the required parameters but has an intuitive motivation: we care only about the existence of a character and not its vertical position.
3. **training:** add an extra shortcut branch, consisting of a single 1D convolution layer, at the output of the CNN backbone. This branch results to an extra character sequence estimation, trained in parallel to the recurrent branch. Both branches use a CTC loss. The motivation behind such a choice comes from the increased difficulty of training recurrent layers. However, if such a straightforward shortcut exists, the output of the CNN backbone should converge to more discriminative features, ideal for fully harnessing the power of recurrent layers compared to an end-to-end training scheme.

The contribution of this paper is best highlighted through the experimental section, where we achieve state-of-the-art results with the aforementioned choices, despite the simplicity of the employed network. Furthermore, other state-of-the-art existing methods propose complex architectures and augmenta-

tion schemes which are orthogonal to our approach, highlighting the importance of the suggested best practices.

2 Related Work

As is the case with most, if not all, tasks in computer vision, modern HTR literature is dominated by neural network-based methods. Recurrent neural networks have become the baseline [15,21], as they naturally fit to the sequential nature of handwriting.

Recurrent-based approaches have thus practically overshadowed the previous state-of-the-art, which was based mostly on Hidden Markov Model (HMM)-based approaches. Since the introduction of the standard recurrent network paradigm [6,7], many key advances have emerged paving the way for very efficient HTR systems. A characteristic example is the integration of the Long Short-Term Memory models (LSTMs) into HTR systems [10], that effectively dealt with the vanishing gradient problem. More importantly, Graves et al. [9] introduced a very effective algorithm for training such HTR systems with sequence-based loss using dynamic programming. Specifically, this Connectionist Temporal Classification (CTC) method and corresponding output layer [8], a differentiable output layer that maps a sequential input into per-time unit softmax outputs, allows simultaneous sequence alignment and recognition with a suitable decoding scheme. Multi-dimensional recurrent networks have been considered for HTR [15], however there has been criticism that the extra computational overhead may not translate to an analogous increase in efficiency [21].

Even though in this work we will focus only on greedy decoding of a CTC-trained network, research on decoding schemes is also active [4], with the beam search algorithm being a popular approach, capable of exploiting an external lexicon as an implicit language model.

Sequence-to-Sequence approaches, involving translating an input sequence to an output sequence of a different length in general, became very popular when achieved state-of-the-art results in Natural Language Processing and gradually evolved to Transformer networks with attention mechanisms [29]. Such approaches were later adopted successfully by the HTR community [3,19,25,27,30].

Recent research directions include complex augmentation schemes [14,16,31], novel network architectures/modules (e.g. Seq2Seq/Transformers, Spatial Transformer Networks [5], deformable convolutions [24]) and multi-task losses with auxiliary training feeds (e.g. n-gram training [28]).

3 Proposed HTR System

In what follows, we will describe in detail the proposed HTR system with emphasis given on the suggested best practice modifications. The described system takes as input either a word or a line image and then returns the predicted sequence of characters based on an unconstrained greedy CTC decoding algorithm [8].

3.1 Preprocessing

The pre-processing steps, applied to every image, are:

1. All images are resized to a resolution of 128×1024 pixels for line images or 64×256 pixels. Initial images are padded (using the image median value, usually zero) in order to attain the aforementioned fixed size. If the initial image is greater than the predefined size, the image is rescaled. The padding option aims to preserve the existing aspect ratio of the text images.
2. During training, image augmentation is performed. A simple global affine augmentation is applied at every image, considering only rotation and skew of small magnitude in order to generate valid images. Additionally, gaussian noise is added to the images.
3. Each word/line transcription has spaces added before and after, i.e. “He rose from” is changed to “ He rose from ”. This operation aims to assist the system to adapt to the marginal spaces that exist in the majority of the images during the training phase. For the testing phase, these additional spaces are discarded.

Augmentation operations are part of every modern deep learning system and can consistently provide increased performance, allowing better generalization [21]. The used augmentation scheme is very basic, trying to have minimal overhead from this step.

The addition of extra spaces in the transcription is not explicitly referred to recent existing works, but it is intuitive given the pad operation of step 1 which creates large empty margins. It has a minor yet positive impact to our system and thus is added as a step. Due to the reduced significance of this step in the overall performance, it is not explored in the experimental section.

On the other hand, we found the padding operation critical in many settings. A typical trade-off, met in many recent text recognition/spotting works, concerns the definition of the input size: using a predefined fixed size can assist the architectural design of CNNs and training time requirements, while retaining the initial image size by processing individually each image (e.g. [26]) may lead to better performance at the cost of discarding the mini-batch option.

Modern DNN training relies on creating batches of several images, since batch manipulation of images can notably affect the training time by fully utilizing the GPU resources. Thus image resizing is a widely-used first step for any vision problem when DNNs are involved. On the contrary, when using different sized images by processing each image individually and update the network’s weights after a predefined number of images, as if a batch was processed, leads to an impractical time-consuming training procedure to otherwise lightweight DNNs, where the existing hardware is under-utilized.

In this work, contrary to the majority of existing approaches, we propose a simple, yet elegant, solution: we aim to retain the aspect ratio of the images and simultaneously organize them into batches. The images are transformed into the same, predefined, shape without resizing, if possible. Specifically, if the image size is smaller than the predefined size, we pad the image accordingly. The

padding operation is performed equally at each direction, positioning the initial image at the center of the new one, with a fixed value, the median value of the initial image. If the image is larger than the predefined size, it is resized, affecting the aspect ratio. To assist the proposed approach, we can compute the average height and width over the whole set of the initial images and select an appropriate size in order to perform the aforementioned resize operation scarcely (only for very large words/sentences) and thus avoid deformations that are generated by frequently violating the aspect ratio.

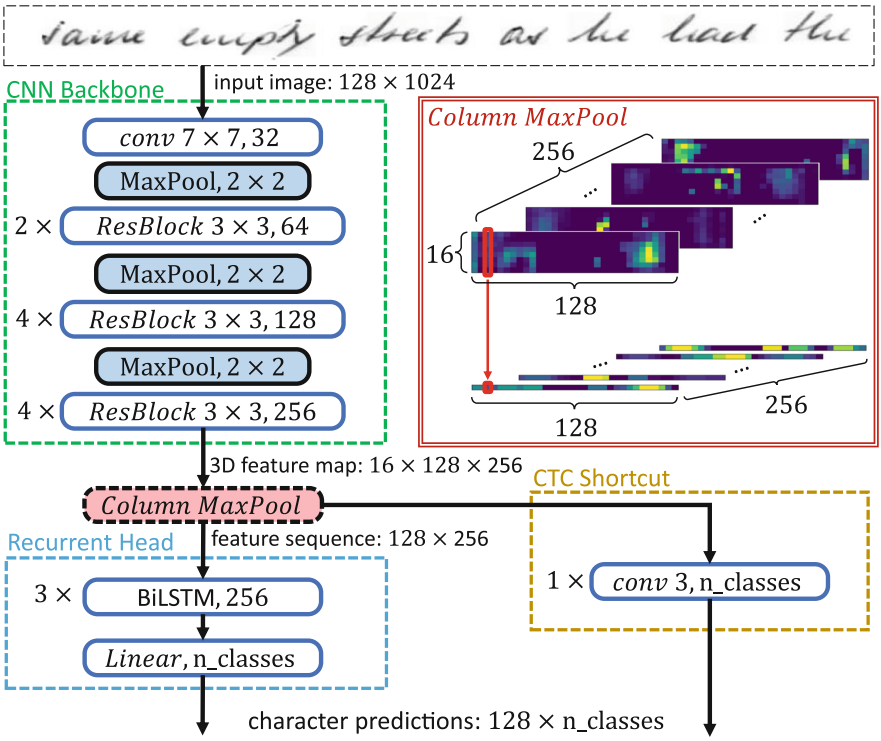

Fig. 1. Overview of the DNN architecture. Apart from the CNN backbone, the recurrent head, we also depict the auxiliary CTC shortcut branch which will be the core component of the proposed training modification. Furthermore, we visualize the column-wise max-pooling operation that is performed between the CNN backbone and the recurrent head.

3.2 Network Architecture

The model that we will use to test the proposed technique can be characterized as a convolutional-recurrent architecture (an architecture overview is depicted in Fig. 1). The convolutional-recurrent architecture can be broadly defined as a

convolutional backbone being followed by a recurrent head, typically connected to a CTC loss. Convolutional-recurrent variants have given routinely very good results for HTR [5,21].

Convolutional Backbone: In our model, the convolutional backbone is made up of standard convolutional layers and ResNet blocks [12], interspersed with max-pooling and dropout layers. In particular, the first layer is a 7×7 convolution with 32 output channels, followed by cascades of 3×3 ResNet blocks [12]: a series of 2 ResNet blocks with 64 output channels, 4 ResNet blocks with 128 output channels and 4 ResNet blocks with 256 output channels. The standard convolution and the ResNet blocks are all followed by ReLU activations, Batch Normalization and dropout. Between cascades of blocks we downscale the produced feature map with 2×2 max-pooling operations of stride 2, as shown in Fig. 1. Overall, the convolutional backbone accepts a line image and outputs a tensor of size $h \times w \times d$ (e.g. assuming the line image case, the tensor is of size $16 \times 128 \times 256$).

Flattening Operation: The convolutional backbone output should be transformed into a sequence of features in order to processed by recurrent networks. Typical HTR approaches, assume a column-wise approach (towards the writing direction) to ideally simulate a character by character processing. In our work, the CNN output is flattened by a max-pooling operation in a column-wise manner. Flattening of the extracted feature maps by the widely-used concatenation operation would result into a sequence of length w with feature vectors of size hd, while max-pooling results to reduced feature vectors of size d. Apart from the apparent computational advantage, *column-wise max-pooling* achieves model translation invariance in the vertical direction. In fact, the reasoning behind max-pooling is that we care only about the existence of features related to a character and not their spatial position. This has been the major motivation for *column-wise max-pooling*, as successfully employed in our previous works [24,25,28].

Recurrent Head: The recurrent component consists of 3 stacked Bidirectional Long Short-Term Memory (BiLSTM) units of hidden size 256. These are followed by a linear projection layer, which converts the sequence to a size equal to the number of possible character tokens, $n_{classes}$ (including the blank character, required by CTC). The final output of the recurrent part can be translated into a sequence of probability distributions by applying a softmax operation. During evaluation, the aforementioned greedy decoding is performed by selecting the character with the highest probability at each step and then removing the blank characters from the resulting sequence [8].

3.3 Training Scheme

The training of the HTR system is performed via an Adam [13] optimizer using an initial learning rate of 0.001 which gradually decreases using a multistep

scheduler. The overall training epochs are 240 and the scheduler decreases the learning rate by a factor of 0.1 at 120 and 180 epochs.

This optimizing scheme, with minor modifications, is commonly used for HTR systems. Nonetheless, we assume an end-to-end training approach where both the convolutional and the recurrent parts of the system are optimized through the final CTC loss. Even though this typical approach produces well-performing solutions, the LSTM head may encumbers the overall training procedure, since recurrent modules are known to exhibit convergence difficulties.

To circumvent this training hindrance, we introduce an auxiliary branch as shown in Fig. 1. We dub this extra module as a "CTC shortcut". In what follows, we describe this module and its functionality in detail.

CTC Shortcut: Architecture-wise, the CTC shortcut module consists only of a single 1D convolutional layer, with kernel size 3. Its output channels equal to the number of the possible character tokens ($n_{classes}$). Therefore, the 1D convolutional layer is responsible for straightforwardly encoding context-wise information and providing an alternative decoding path. Note that we strive for simplicity for this auxiliary component, since its aim is to assist the training of the main branch and thus a shallow convolutional part of only one layer is ideal for this task. We do not expect from the CTC branch to result to precise decodings.

The CTC shortcut is trained along with the main architecture using a multi-task loss by adding the corresponding CTC losses of the two branches with the appropriate weights. Specifically, if f_{cnn} represents the convolutional backbone, f_{rec} represents the recurrent part and $f_{shortcut}$ represents the proposed shortcut branch, while I is an input image and s its corresponding transcription, the multi-task loss is written as:

$$L_{CTC}(f_{rec}(f_{cnn}(I)); s) + 0.1\, L_{CTC}(f_{shortcut}(f_{cnn}(I)); s) \tag{1}$$

Since CTC shortcut acts only as an auxiliary training path, it is weighted by 0.1 to reduce its relative contribution to the overall loss.

The motivation behind this extra branch is rather simple: overall convergence is assisted by quickly generating discriminative features at the top of the CNN backbone through the straightforward 1D convolutional path, simplifying the training task for the recurrent part. Due to its training-oriented assisting nature, CTC shortcut is used only during training and omitted during evaluation. Therefore, this proposed shortcut does not introduce any overhead during inference.

4 Experimental Evaluation

Evaluation of the proposed system is performed on two widely used datasets, IAM [18] and Rimes [11]. The ablation study, considering different settings of

the proposed methodology, is performed on the challenging IAM dataset, consisting of handwritten text from 657 different writers and partitioned into writer-independent train/validation/test sets (we use the same set partition as in [21]). All experiments follow the same setting: line-level or word-level recognition using a lexicon-free unconstrained greedy CTC decoding scheme. Character Error Rate (CER) and Word Error Rate (WER) metrics are reported in all cases (lower values are better).

4.1 Ablation Study

First, we explore the impact of the proposed modifications over both the validation and the test set of IAM dataset. Moreover, both line-level recognition (Table 1) and word-level recognition (Table 2) are considered. Specifically, we investigate the difference in performance when we: 1) use *resized* or *padded* (retain aspect-ratio case) input images, 2) use *concatenation* of *max-pooling* flattening operation between the convolutional backbone and the recurrent head and 3) use or not the *CTC shortcut during* the training process.

Table 1. Line-level recognition results for IAM dataset: exploring the impact of the proposed modifications.

Preprocessing	Flattening	CTC shortcut	Validation		Test	
			CER (%)	WER (%)	CER (%)	WER (%)
Resized	Concatenation	No	4.28	15.29	5.93	19.57
		Yes	3.72	13.18	5.11	16.96
Resized	Max-pooling	No	3.73	13.54	5.28	17.77
		Yes	3.47	12.77	4.85	16.19
Padded	Concatenation	No	4.06	14.40	5.54	18.60
		Yes	3.37	12.22	4.71	15.94
Padded	Max-pooling	No	3.46	12.55	4.93	16.81
		Yes	**3.21**	**11.89**	**4.62**	**15.89**

The following observations can be made:

- Retaining the aspect-ratio of the images (padded option) achieves improved results for the majority of cases.
- Performing the flattening operation via max-pooling not only is more cost-effective, but it has a positive impact on performance. This is more evident in line-level recognition setting.
- Training with a CTC shortcut module provides notable boost over all cases. For example, in line-level recognition the significant difference in performance when considering different flattening operations is considerably decreased when the CTC shortcut approach is adopted (e.g. for padded line-level recognition the WER performance difference drops from 1.79% to only 0.05%). This hints that the initial difference in performance is mainly attributed to difficulties in training (concatenated version has a much larger feature vector to manage). Note that evaluating the CTC shortcut branch yields poor

Table 2. Word-level recognition results for IAM dataset: exploring the impact of the proposed modifications.

Preprocessing	Flattening	CTC shortcut	Validation		Test	
			CER (%)	WER (%)	CER (%)	WER (%)
Resized	Concatenation	No	4.35	12.55	5.58	15.46
		Yes	4.27	12.02	5.46	15.13
Resized	Max-pooling	No	4.25	12.17	5.69	15.87
		Yes	4.09	11.65	5.23	14.40
Padded	Concatenation	No	4.17	11.99	5.66	15.66
		Yes	3.98	11.50	5.37	14.98
Padded	Max-pooling	No	4.00	11.25	5.43	15.06
		Yes	**3.76**	**10.76**	**5.14**	**14.33**

decodings, despite the notable performance increase of the main network. For example, assuming line-level recognition and the padded/max-pooling setting, we report 5.26% CER/19.76% WER for the validation set and 7.36% CER/25.66% WER for the test set.

– Applying all three modifications together achieves the best results across all setting and metrics.
– Word recognition reports improved results compared to line-level recognition with respect to the WER metric. This was expected, since word-level setting assumes perfect word segmentation. Interestingly enough, this is not the case for the CER metric. This can be explained by the lack of sufficient context (i.e. find a capital letter or a punctuation from the whole line information).

We also explore in more depth the CTC shortcut option, which seems to provide the best boost in performance. Specifically, we report the progress of both the loss and the CER/WER metrics (over the validation set) during the training procedure for the line-level recognition setting. The loss curves are depicted in Fig. 2, while the validation set evaluation metrics are reported in Fig. 3. As we can see, loss curves are similar, but the case of CTC shortcut consistently has slightly better behavior. The impact of the CTC shortcut is more clearly shown in CER/WER curves and thus solutions with greater generalization properties are expected when a model is trained along with the CTC shortcut.

4.2 Comparison to State-of-the-Art Systems

Finally, we compare our method to several existing state-of-the-art methods, as shown in Table 3. The reported methods follow the same setting: line-level lexicon-free recognition. The proposed HTR system along with the suggested modifications achieves results comparable to the best performing methods. Notably, it outperforms the majority of existing works for both datasets and metrics despite many of the reported methods propose novel elements to further increase performance that are in general orthogonal to our approach. For example, the work of Chowdhury et al. [3] presents better WER for the RIMES dataset while using a sequence-to-sequence approach (such models can produce

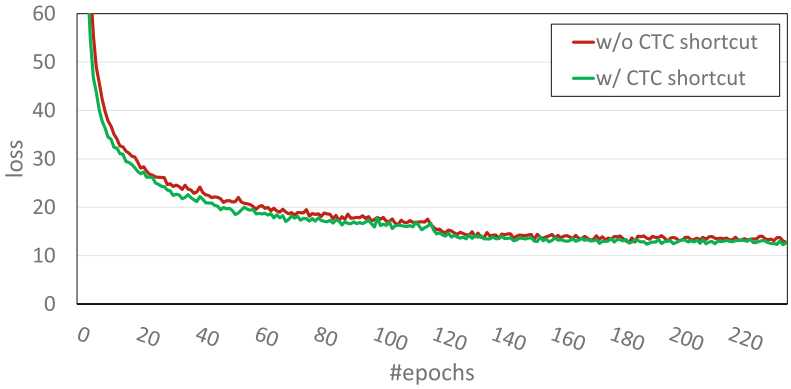

Fig. 2. Behavior of HTR performance in terms of loss value with and without the extra CTC shortcut branch during the training phase. Reported curves correspond to the proposed line-level HTR system trained on the IAM dataset.

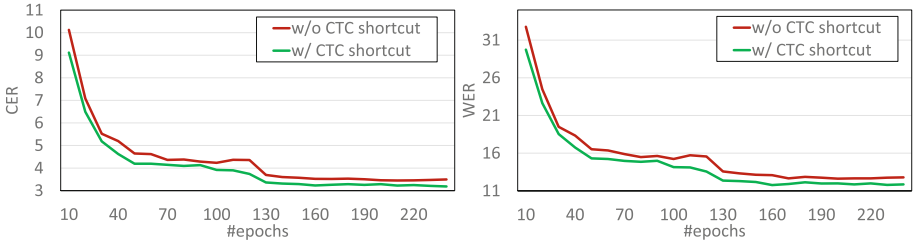

Fig. 3. Behavior of HTR performance in terms of CER (left) and WER (right) metrics with and without the extra CTC shortcut branch during the training phase. Reported curves correspond to the proposed line-level HTR system, trained on IAM dataset and evaluated on the validation set.

increased WER as implicit language models can be learnt [25]), while our previous work [24] achieves better CER for the IAM dataset while using similar network (max-pooling flattening and padded input images) along with deformable convolutions and a post processing uncertainty reduction algorithm.

Moreover, the very recent work of Luo et al. [16] manages to outperform our method for the word-level recognition setting on the IAM dataset by using a STN component and a complex augmentation method, where "optimal" augmentations are learnt. Specifically, our method achieves 5.14% CER/14.33%, while Luo et al. achieve 5.13% CER/13.35% WER for the exact same setting. Nonetheless, their initial baseline network, stripped of all the extra modules (which could be added to the proposed architecture without any problem), performs poorly: 7.39% CER and 19.12% WER.

Overall, we achieve very competitive results (outperforming other existing lexicon-free methods for line-level recognition on IAM) by only using a typical convolutional-recurrent architecture along with a set of simple, yet intuitive and

effective modifications, forming an effective set of best practice suggestions which can be applied to the majority of HTR systems.

Table 3. Performance comparison for IAM/RIMES datasets (line-level recognition)

Method	IAM		RIMES	
	CER (%)	WER (%)	CER (%)	WER (%)
Chen et al. [2]	11.15	34.55	8.29	30.5
Pham et al. [20]	10.8	35.1	6.8	28.5
Khrishnan et al. [14]	9.78	32.89	–	–
Chowdhury et al. [3]	8.10	16.70	3.59	9.60
Puigcerver [21]	6.2	20.2	2.60	10.7
Khrishnan et al. [14]	9.78	32.89	–	–
Markou et al. [17]	6.14	20.04	3.34	11.23
Dutta et al. [5]	5.8	17.8	5.07	14.7
Wick et al. [30]	5.67	–	–	–
Michael et al. [19]	5.24	–	–	–
Tassopoulou et al. [28]	5.18	17.68	–	–
Yousef et al. [32]	4.9	–	–	–
Retsinas et al. [24]	4.55	16.08	3.04	10.56
Proposed	4.62	15.89	2.75	9.93

5 Conclusions

In this paper, we proposed a series of best practice modifications over typical convolutional-recurrent networks trained with CTC loss. Apart from presenting a fairly compact architecture based on residual blocks, we present three impactful modifications: 1) retain aspect-ratio of input images gathered in batches through a padding operation, 2) apply a column-wise max-pooling operation between the convolutional backbone and the recurrent head of a typical HTR architecture for reduced computational effort and increased performance and 3) enhance performance through a CTC shortcut during training in order to circumvent an end-to-end training over recurrent networks, which have been proven "difficult" to train in various settings. All proposed modifications have proven to be very helpful, considerably increasing the performance of the vanilla network. Overall, the proposed system achieves results in the ballpark of state-of-the-art, while being orthogonal to the majority of modern deep learning modules and approaches.

Acknowledgement. This research has been partially co - financed by the EU and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the calls: "RESEARCH - CREATE - INNOVATE", project *Culdile* (code T1EΔK - 03785) and "OPEN INNOVATION IN CULTURE", project *Bessarion* (T6YBΠ - 00214).

References

1. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer, Heidelberg (2006)
2. Chen, Z., Wu, Y., Yin, F., Liu, C.L.: Simultaneous script identification and handwriting recognition via multi-task learning of recurrent neural networks. In: 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), vol. 1, pp. 525–530. IEEE (2017)
3. Chowdhury, A., Vig, L.: An efficient end-to-end neural model for handwritten text recognition (2018)
4. Collobert, R., Hannun, A., Synnaeve, G.: A fully differentiable beam search decoder. In: International Conference on Machine Learning, pp. 1341–1350. PMLR (2019)
5. Dutta, K., Krishnan, P., Mathew, M., Jawahar, C.: Improving CNN-RNN hybrid networks for handwriting recognition. In: 2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR), pp. 80–85. IEEE (2018)
6. Fischer, A., Keller, A., Frinken, V., Bunke, H.: Lexicon-free handwritten word spotting using character HMMs. Pattern Recogn. Lett. **33**(7), 934–942 (2012)
7. Fischer, A.: Handwriting recognition in historical documents. Ph.D. thesis, Verlag nicht ermittelbar (2012)
8. Graves, A.: Connectionist temporal classification. In: Graves, A. (ed.) Supervised Sequence Labelling with Recurrent Neural Networks. Studies in Computational Intelligence, vol. 385, pp. 61–93. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-24797-2_7
9. Graves, A., Fernández, S., Gomez, F., Schmidhuber, J.: Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In: Proceedings of the 23rd International Conference on Machine Learning, pp. 369–376 (2006)
10. Greff, K., Srivastava, R.K., Koutník, J., Steunebrink, B.R., Schmidhuber, J.: LSTM: a search space odyssey. IEEE Trans. Neural Netw. Learn. Syst. **28**(10), 2222–2232 (2016)
11. Grosicki, E., Carre, M., Brodin, J.M., Geoffrois, E.: Rimes evaluation campaign for handwritten mail processing (2008)
12. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
13. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. In: Proceedings of the International Conference on Learning Representations (ICLR) (2015)
14. Krishnan, P., Dutta, K., Jawahar, C.: Word spotting and recognition using deep embedding. In: 2018 13th IAPR International Workshop on Document Analysis Systems (DAS), pp. 1–6. IEEE (2018)
15. Leifert, G., Strau, T., Gr, T., Wustlich, W., Labahn, R., et al.: Cells in multidimensional recurrent neural networks. J. Mach. Learn. Res. **17**(97), 1–37 (2016)
16. Luo, C., Zhu, Y., Jin, L., Wang, Y.: Learn to augment: joint data augmentation and network optimization for text recognition. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 13746–13755 (2020)
17. Markou, K., et al.: A convolutional recurrent neural network for the handwritten text recognition of historical Greek manuscripts. In: Del Bimbo, A., et al. (eds.) ICPR 2021. LNCS, vol. 12667, pp. 249–262. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-68787-8_18

18. Marti, U.V., Bunke, H.: The IAM-database: an English sentence database for offline handwriting recognition. Int. J. Doc. Anal. Recogn. **5**(1), 39–46 (2002)
19. Michael, J., Labahn, R., Grüning, T., Zöllner, J.: Evaluating sequence-to-sequence models for handwritten text recognition. In: 2019 International Conference on Document Analysis and Recognition (ICDAR), pp. 1286–1293. IEEE (2019)
20. Pham, V., Bluche, T., Kermorvant, C., Louradour, J.: Dropout improves recurrent neural networks for handwriting recognition. In: 2014 14th International Conference on Frontiers in Handwriting Recognition, pp. 285–290. IEEE (2014)
21. Puigcerver, J.: Are multidimensional recurrent layers really necessary for handwritten text recognition? In: 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), vol. 1, pp. 67–72. IEEE (2017)
22. Retsinas, G., Sfikas, G., Gatos, B.: Transferable deep features for keyword spotting. In: Multidisciplinary Digital Publishing Institute Proceedings, vol. 2, p. 89 (2018)
23. Retsinas, G., Sfikas, G., Nikou, C.: Iterative weighted transductive learning for handwriting recognition. In: Lladós, J., Lopresti, D., Uchida, S. (eds.) ICDAR 2021. LNCS, vol. 12824, pp. 587–601. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-86337-1_39
24. Retsinas, G., Sfikas, G., Nikou, C., Maragos, P.: Deformation-invariant networks for handwritten text recognition. In: 2021 IEEE International Conference on Image Processing (ICIP), pp. 949–953. IEEE (2021)
25. Retsinas, G., Sfikas, G., Nikou, C., Maragos, P.: From Seq2Seq recognition to handwritten word embeddings. In: Proceedings of the British Machine Vision Conference (BMVC) (2021)
26. Sudholt, S., Fink, G.A.: PHOCNet: a deep convolutional neural network for word spotting in handwritten documents. In: Proceedings of the 15th International Conference on Frontiers in Handwriting Recognition (ICFHR), pp. 277–282 (2016)
27. Sueiras, J., Ruiz, V., Sanchez, A., Velez, J.F.: Offline continuous handwriting recognition using sequence to sequence neural networks. Neurocomputing **289**, 119–128 (2018)
28. Tassopoulou, V., Retsinas, G., Maragos, P.: Enhancing handwritten text recognition with n-gram sequence decomposition and multitask learning. In: 2020 25th International Conference on Pattern Recognition (ICPR), pp. 10555–10560. IEEE (2021)
29. Vaswani, A., et al.: Attention is all you need. In: Advances in Neural Information Processing Systems, pp. 5998–6008 (2017)
30. Wick, C., Zöllner, J., Grüning, T.: Transformer for handwritten text recognition using bidirectional post-decoding. In: Lladós, J., Lopresti, D., Uchida, S. (eds.) ICDAR 2021. LNCS, vol. 12823, pp. 112–126. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-86334-0_8
31. Wigington, C., Stewart, S., Davis, B., Barrett, B., Price, B., Cohen, S.: Data augmentation for recognition of handwritten words and lines using a CNN-LSTM network. In: 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), vol. 1, pp. 639–645. IEEE (2017)
32. Yousef, M., Hussain, K.F., Mohammed, U.S.: Accurate, data-efficient, unconstrained text recognition with convolutional neural networks. Pattern Recogn. **108**, 107482 (2020)